Evaluation of a Framework for Dynamic Source Selection in Stream Processing

著者	Ohki Kosuke, Watanabe Yousuke, Kitagawa Hiroyuki
内容記述	2009 International Conference on Complex, Intelligent and Software Intensive Systems : Fukuoka,Japan ; March 16-March 19, 2009
journal or publication title	International Conference on Complex, Intelligent and Software Intensive Systems
page range	1106-1111
year	2009
URL	http://hdl.handle.net/2241/106175

doi: 10.1109/CISIS.2009.167

Evaluation of a Framework for Dynamic Source Selection in Stream Processing

Kosuke OHKI
Graduate School of Systems and Information Engineering, University of Tsukuba
ohki@kde.cs.tsukuba.ac.jp

Yousuke WATANABE
Global Scientific Information and Computing Center, Tokyo Institute of Technology
watanabe@de.cs.titech.ac.jp

Hiroyuki KITAGAWA
Graduate School of Systems and Information Engineering, University of Tsukuba
kitagawa@cs.tsukuba.ac.jp

Abstract

The volume of stream data delivered from different information sources is increasing. There are a variety of demands to utilize such stream data for applications. Stream processing systems can continuously process stream data according to user requests. In conventional frameworks, the user must explicitly specify information sources in advance, and the user cannot change information sources during query processing. However, there are many cases in which target information sources the user is most interested in change over time. We therefore need an additional framework to deal with changes of the target information sources in the same query. We propose dynamic source selection. Our proposal consists of two functions: The first is a new operator to switch information sources dynamically. The second is connection management to reduce connections to unnecessary information sources. We have implemented the proposed framework in a stream processing system and have evaluated the efficiency of our proposed method in stream data simulator.

1 Introduction

Recent developments in network technology and sensor device technology have made it easy to get stream data. Beyond that, the demand for query processing for stream data has been increasing. Continuous query is a query processing framework for stream data. Stream processing systems [2, 3] process continuous queries. The system continuously generates the query result when new stream data arrives. Most existing continuous query languages are based on SQL.

Previous stream processing systems only process stream data from information sources which users explicitly specify in a FROM clause of a query. The system cannot dynamically change the target information sources during query processing.

However, we face many situations when users want the system to change the target information sources during query processing. For example, we suppose a request to track a moving objects in an environment that includes many network cameras and location sensors which is taken by moving objects. The user wants to receive video streams from cameras near the tracking object. But, the user cannot code a query corresponding to the request. Because the target cameras change according to the current position of the tracking object.

We therefore need a framework that the system can dynamically select the target information sources during query processing. We propose a scheme for dynamic selection of information sources. Our contributions in this paper are as follows:

- **Target Selective Join operator (TS-Join operator) to switch information sources dynamically**. We define a new operator "TS-Join". TS-Join selects the target information sources depending on a value of input data.
- **Connection management**. Connecting to information sources requires a lot of system resources. We want to reduce connections to needless information sources. Therefore, we provide functions to manage connections dynamically.
- **Implementing the proposed framework into our stream processing system.** We have implemented dynamic source selection in the stream processing system [9] we developed.
- **Experiment in stream data simulator.** We constructed experimental environment. The test scenario applies the tracking of video data on a moving objects. We make sure the proposed method is efficient in stream data simulator.

[4] is our preliminary work. It does not contain experimental evaluations. This paper, on the other hand, includes more

978-0-7695-3575-3/09 $25.00 © 2009 IEEE
DOI 10.1109/CISIS.2009.167

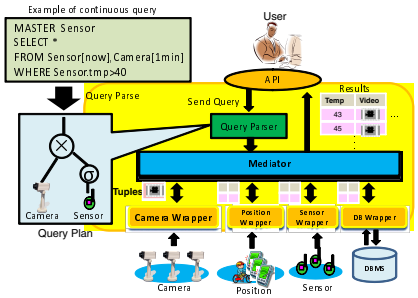

Figure 1. Stream Processing System

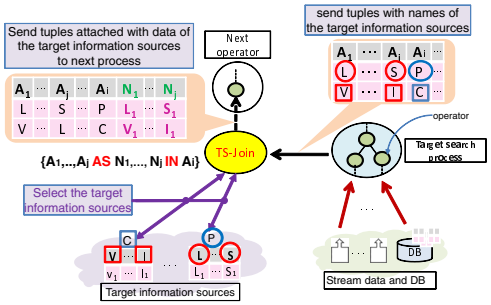

Figure 2. Basic overview of dynamic source selection

detail on our proposed framework and intensive experimental evaluations.

The remainder of this paper is organized as follows: Section 2 presents an architecture of stream processing system. Section 3 describes the dynamic source selection framework. Section 4 describes our implementation system. Section 5 describes our experiment in stream data simulator. Section 6 introduces related work. Section 7 concludes the paper, and mentions the future research issues.

2 Stream Processing System

In this paper, we assume a stream processing system based on the architecture in Fig.1. A wrapper is a module which transforms stream data from information source into internal data representation. We employ a relational model as the internal data model. A wrapper sends the data to a mediator. When a mediator receives the data, it evaluates operators according to a query plan. The query plan is generated by the query parser. Query results generated by a mediator are immediately delivered to the users.

Continuous queries are written in SQL-like query language. The query in Fig.1 is an example of continuous query. It means "Deliver integration result of video and sensor data when a value of temperature sensor becomes greater than 40". A MASTER clause specifies events to trigger query processing. When new data arrives from the streams written in the MASTER clause, the query is evaluated by the system. A window slides as time progresses, and tuples within the range are evaluated when each event occurs. "now" in the FROM clause of Fig.1 indicates the most recent tuples, and "1mim" indicates tuples within the most recent minute.

3 Dynamic Source Selection

In our research, we want the system dynamically selects the target information sources. To handle changes of the target information sources, we propose dynamic source selection. Basic overview of the dynamic source selection is Fig.2. First, the system searches the current target information sources, then the system selects the stream data from the target information sources. The components are as follows:

- **Target search process** searches names of target information sources. The process is composed of an arbitrary relational algebra.
- **TS-Join operator** can dynamically select information sources depending on the value of input data and the operator generates the result of the Cartesian product.

In Fig.2, target search process searches target information sources in stream data and DB. And, the target search process outputs the tuple which contains the target information sources to TS-Join operator. When TS-Join operator receives the tuple, the operator selects the target information sources corresponding to the tuple from target search process.

Before the TS-Join operator selects the target information sources, the system must establish connections to the sources. Therefore, we implemented connection management. Connection management controls connections to necessary information sources.

For details, we explain TS-Join operator in section 3.1 and connection management in section 3.2.

3.1 TS-Join operator

3.1.1 Definition

We define the new operator TS-Join, which can switch the target information sources according to a input value. Definition of the TS-Join operator for a table $R(A_1, \ldots, A_n)$ follows:

$$TS\text{-}Join_{A_i, S_A, S_N}(R)$$

$$= \bigcup_{r \in R} \{r \times t | t \in \delta_{N_1,\ldots,N_j} \pi_{attr(r.A_1,\ldots,r.A_j)}(relation(r.A_i))$$

Here r is a tuple of R. S_A and S_N mean $\{A_1, \ldots, A_j\}$ and $\{N_1, \ldots, N_j\}$ respectively. $relation(r.A_i)$ is a relation corresponding to the value of A_i in tuple r, and $attr(r.A_1, \ldots, r.A_j)$ represents a set of attribute names corresponding to values of $A_1, \ldots, A_j$. $\delta_{N_1,\ldots,N_j}$ is a rename operator to rename attribute names to $N_1, \ldots, N_j$. The three parameters of the TS-Join operator are as follows:

- A_i is an attribute containing the name of the information source $I(B_1, \ldots, B_m)$ the user wants to select
- S_A is a set of R's attribute S specifying a set of I's attributes the user wants to get
- S_N represents a set of attribute names to rename the attributes obtained by the TS-Join operator

We explain the behavior of the TS-Join operator. The TS-Join operator can select which information sources are extracted by referring to the value of attributes in tuple r. First, according to the value of $r.A_i$, the TS-Join operator finds the relation whose name matches this value. Second, the TS-Join operator drops the attributes not corresponding to the values of $r.A_1, ..., r.A_j$. Third, it renames the remaining attributes to $N_1, \ldots, N_j$. It then computes the Cartesian product of r and the obtained data.

3.1.2 Syntax of TS-Join

The following explains the syntax of TS-Join in a query that uses the TS-Join operator. The TS-Join is coded as follows:

$$TS\ JOIN\ \ A_1[, \ldots, A_j]\ \ AS\ \ N_1[, \ldots, N_j]\ \ IN\ \ A_i$$

$A_1, \ldots, A_j$, A_i, and $N_1, \ldots, N_j$ are the same in equation of Section 3.1.1.

3.1.3 Application of dynamic source selection

We consider an application for tracking video data of moving objects in Fig.3. We suppose there are many network cameras and a database which contains locations of cameras and position sensors with moving objects. Camera i ($1 \leq i \leq N$) is a video stream from i-th camera. It contains the time stamp and video data at the time. CamLoc contains names of cameras, locations of cameras and attributes of the video data. Position is the stream from position sensors taken by moving objects.

The query in Fig.3 means that the user wants to receive video streams from network cameras whose distances from the tracking object "A" are within five meters. The target information sources in this example are a set of cameras near the tracking object. Relevant camera names are obtained by integrating the location of cameras with the position of the tracking object "A" (Target search process).

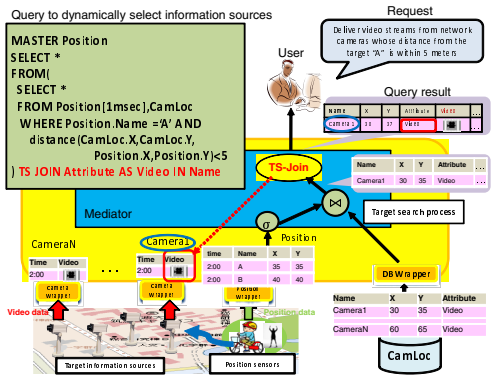

Figure 3. Application of dynamic source selection

Arrows in Fig.3 express data flow and circles are operators (Selection, Join and TS-Join). When the system receives the position of the tracking object, it finds names of cameras near the tracking object. At this time, the system gets a tuple with the name attribute "Camera1". The TS-Join operator then selects Camera1 from information sources and gets the video data from "Video" in Camera1. The TS-Join operator outputs a tuple with the Video attribute. The system repeats the process in the same way when the position data of the tracking object arrives.

3.2 Connection Management

The system must connect to the target information sources before the TS-Join operator selects the sources. However, it doesn't make sense to connect to unnecessary information sources. We need to dynamically connect to necessary information sources. Therefore, we implement connection management.

To use the connection management, we must specify a query. When the system receives names of information sources, the system creates wrappers corresponding to the names. Then, the system can receive stream data from the information sources.

For example, in Fig.3, the system gets video data from cameras whose distances from the tracking object are within five meters. The system can create connections to cameras before evaluating the TS-Join operator by connection management. When the tracking object moves away, the connected cameras are not needed and the system can release them. Connection management consists of two functions to manage connections by queries.

- Create connection: This query is used to make the

```
MASTER Position
ACTIVATE CamLoc.Name
FROM  Position[now],CamLoc
WHERE Position.Name='A'
AND distance(CamLoc.X,CamLoc.Y,Position.X,Position.Y)<10
```

Figure 4. Query to create connections

```
MASTER  Position
DEACTIVATE CamLoc.Name
FROM Position[now],CamLoc
WHERE Position.Name='A'
AND distance(CamLoc.X,CamLoc.Y,Position.X,Position.Y)>=10
```

Figure 5. Query to release connections

system connect to necessary information sources. A query example is shown in Fig.4. It means that "when the tracking object approaches cameras within ten meters, the system connects to the cameras." ACTIVATE means that the connection management can activate a wrapper at the mediator module according to the name sets of the target informations that result from the query processing. Here, we set the threshold to ten meters rather than five meters, because there is a delay before starting to receive stream data.

- Release connection: The system can reduce system load by releasing unnecessary connections. A query example is shown in Fig.5. DEACTIVATE means the connection management can deactivate unnecessary connections in mediator.

When a query for create connection is triggered, a mediator indicates a relevant wrapper to connect to the necessary information source, and the wrapper starts to receive input data and convert the data into tabular form.

4 Implementation

We have implemented the proposed framework in our stream processing system[9, 4]. The system uses Java SDK 1.6.0_03 and JMF(Java Media Frameworks)[5] to process video stream in stream processing system. A camera wrapper uses QuickTime for Java[6] to handle video stream.

We have also implemented a tracking application on our stream processing system. Fig.6 is a screenshot of the system. A list of connected information sources appears at the leftside of Fig.6. The system selects the video stream from the target cameras in all cameras by the dynamic source selection.

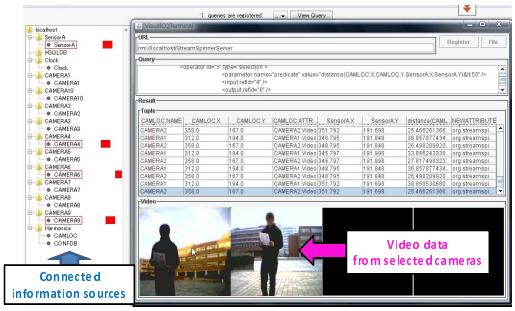

Figure 6. Screenshot of stream processing system and tracking application

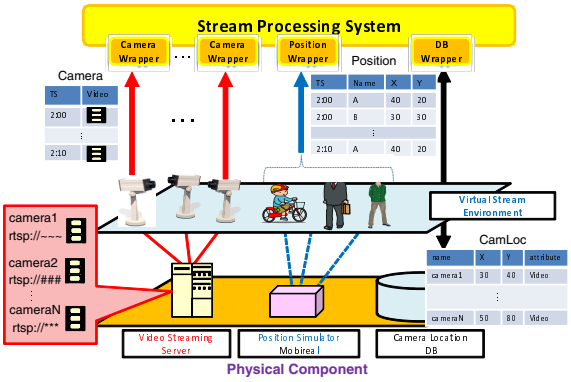

Figure 7. Stream data simulator

5 Evaluation experiment

5.1 Stream data simulator

This section explains the simulator for grand-scale experiments. An overview of the simulator is shown in Fig.7 The simulator delivers stream data from camera and position sensors in virtual two dimension map to the stream processing system. The simulator consists of video streaming server, camera location DB, and position simulator. We explain the composition in detail as follows.

- **Video streaming server**. The server delivers video data. Each item of video data is given a unique camera name and URL to access the video. When our system accesses the URL, the server delivers the video stream corresponding to the URL. We use DarwinStreamingServer[8].

- **Camera location DB**. The DB stores all locations of cameras in the virtual map. The DB contains a camera name, x-axis and y-axis of cameras location, and an attribute name containing video data.

- **Position data of moving objects**. We use mobireal[7] to simulate the position data.

The specifications for the machine running the video streaming server is OS: MacOSX10.4.11, CPU: 1.25 GHz PowerPC G4, Memory: 512MB. The specifications for the machine running the stream processing system is OS: windows Vista, CPU: 2.00GHz AMD Athlon 64 2 Dual Core Processor 3800, Memory: 2GB.

5.2 Evaluation experiment in stream data simulator

We experimented in stream data simulator to prove efficiency of the proposed framework. The experiments are as follows:

1. Compare system loads of the proposed framework and a naive approach. The naive approach here means the system connects to all available cameras.

2. Investigate CPU load and memory usage when tracking the number of moving objects varies.

3. Make sure the proposed framework is efficient with many cameras and moving objects (# of camera is 100 and # of moving objects is 1000).

5.2.1 Comparison of the proposed framework and naive approach

We prepared ten cameras and position data for a thousand moving objects in stream data simulator. Queries for the experiments are as follows:

1. Naive approach: We register a query whose FROM clause contains names of 10 cameras. The query is showed in Fig.8. The system connects to all cameras and continuously selects video data from cameras whose distance from the tracking object are within 50 meters.

2. The proposed approach: We register three queries to use dynamic source selection framework. The query in Fig.3 gets video data by TS-Join operator. The queries in Fig.4 and Fig.5 are used to invoke connection management functions. We set the threshold of distance in Fig.3, Fig.4 and Fig.5 to 50, 60, and 60 respectively.

```
MASTER Position
SELECT *
FROM(
    SELECT Camera1.Video
    FROM Position[1msec],Camera1[1min],CameLoc
    WHERE Position.Name='A'
      AND   CamLoc.Name = 'Camera1'
      AND   distance(CamLoc.X,CamLoc.Y,Position.X,Position.Y)<50
    UNION
    .
    .
    .
    UNION
    SELECT Camera10.Video
    FROM Position[1msec],Camera10[1min],CamLoc
    WHERE Position.Name='A'
      AND   CamLoc.Name='Camera10'
      AND   distance(CamLoc.X,CamLoc.Y,Position.X,Position.Y)<50
    )AS AllCamera
```

Figure 8. Query to connect to all cameras

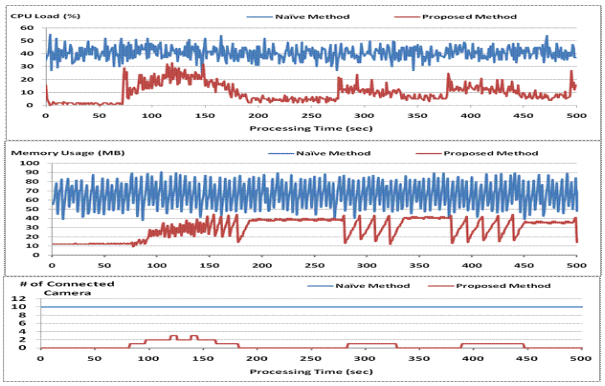

Figure 9. CPU Load and Memory Usage and number of connected cameras in comparison of naive and proposed approaches

We measured CPU load and memory usage for two approaches. The results are in Fig.9. The horizontal axis indicates simulation time and vertical axes indicate CPU load and memory usage and number of connected cameras. The proposed approach is better than the naive approach. When the system creates connections to cameras and processes their video data at the range of [80,200],[270,330],[380,460], the CPU load and memory usage increases. However, when the system releases the connections, CPU load falls. The result indicates that the proposed framework can save system resources.

5.2.2 tracking a number of moving objects

Users can track a number of moving objects at one time by the proposed method. The results are shown in Fig.10. The horizontal axis indicates the number of tracking objects and vertical axes indicate CPU load and memory usage. The Fig.10 show that CPU load and memory usage

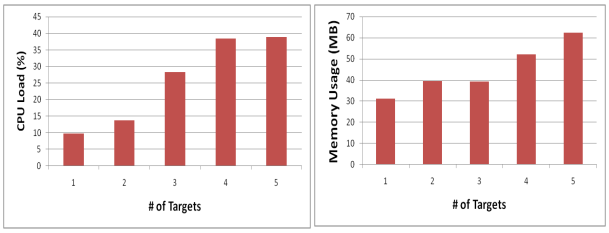

Figure 10. CPU Load and Memory Usage in a number of tracking objects

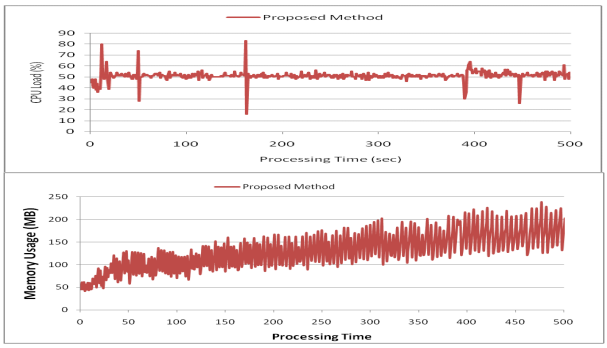

Figure 11. CPU Load and Memory Usage in 100 cameras environment

increase as the number of tracking objects increases. The system can track multiple tracking objects at one time. But, we need sufficient system resources to track more moving objects. Distributed stream processing is one of the solutions to avoid resource problem. Dynamic source selection in distributed environment is a future research issue.

5.2.3 Experiment in the environment with large number of cameras

In this experiment, we prepared position data of a thousand moving objects and a hundred cameras. We use the same queries in Fig4, Fig5 and Fig3. We measured CPU load and memory usage. The results are Fig.11. CPU load increases when the system creates/releases connections. Otherwise, CPU load is constant. We proved the proposed framework is efficient.

6 Related Work

Aurora[2] and STREAM[3] are stream processing systems that can help users to easily create applications to process stream data. When users write a information sources in FROM clause, the system processes only the stream data from the stipulated information sources in the query.

FISQL[1] is a query language for schema conversion in DBMS. FISQL allows a user to write variables in the FROM clause, which means information sources referred to by a query are not fixed. But, FISQL does not focus on stream data.

7 Conclusion

We proposed dynamic source selection to handle change of target information sources, and implement the proposed method in our stream processing system. In order to evaluate the proposed method, we made a virtual numerous stream environment based on tracking application. We proved that efficiency of dynamic source selection in our experiments.

Future research issues exist. We plan to extend our dynamic source selection to fit distributed stream processing environments.

Acknowledgments

This research has been supported in part by Japan Science and Technology Agency (CREST), the Grant-in-Aid for Scientific Research from JSPS (#1820005).

References

[1] Catharine M. Wyss, et al. “Extending Relational Query Optimization to Dynamic Schemas for Information Integration in Multidatabases”, SIGMODf07, 2007.

[2] Daniel J. Abadi, et al. “Aurora:A New Model and Architecture for Data Stream Management”, VLDB Journal Vol.12, No2, pp.120-139, 2003.

[3] Rajeev Motwani, et al. “Query Processing, Resource Management, and Approximation in a Data Stream Management System”. Proc. CIDR, 2003.

[4] Yousuke Watanabe, et al. “A video stream management system for heterogeneous information integration environment”. ICUIMC, 2008.

[5] JMF, http://java.sun.com/products/java-media/jmf/

[6] QuickTime Player, http://www.apple.com/jp/quicktime/

[7] mobireal, http://www.mobireal.net/index-j.html

[8] Darwin Streaming Server, http://www.apple.com/jp/quicktime/streamingserver

[9] http://www.streamspinner.org/